Nick Cobb

Aero 430 – April 28, 2017

1-d & 2-d Finite Element Method Solutions by Assembly

Contents

[Part I: One Dimensional Finite Element Solution 2](#_Toc480913678)

[1-D Problem: 3](#_Toc480913679)

[Approximated Solution (Compact 2nd Order): 4](#_Toc480913680)

[Approximated Solution (Compact 4th Order): 5](#_Toc480913681)

[Modifying for Assembly: 6](#_Toc480913682)

[Element: 6](#_Toc480913683)

[Ghost Value: 6](#_Toc480913684)

[Post-Processing: 8](#_Toc480913685)

[1st Derivative: 8](#_Toc480913686)

[Error: 9](#_Toc480913687)

[Richardson Extrapolation: 9](#_Toc480913688)

[Results: 10](#_Toc480913689)

[Negative Case: 10](#_Toc480913690)

[Positive Case: 18](#_Toc480913691)

[Part II: 2-Dimensional Finite Element Solutions 26](#_Toc480913692)

[Problem: 27](#_Toc480913693)

[Exact Solution for Negative Case: 27](#_Toc480913694)

[Exact Solution for Positive Case: 29](#_Toc480913695)

[2nd Order Approximation: 29](#_Toc480913696)

[Fourth Order Solution(Negative): 30](#_Toc480913697)

[Fourth Order Solution(Positive) 31](#_Toc480913698)

[Mesh and Elements: 32](#_Toc480913699)

[Ghost Points(2nd Order) 33](#_Toc480913700)

[Ghost Points(4th Order) 35](#_Toc480913701)

[Post-Processing: 37](#_Toc480913702)

[Error: 37](#_Toc480913703)

[Richardson Extrapolation: 37](#_Toc480913704)

[Results: 38](#_Toc480913705)

[Negative Case: 38](#_Toc480913706)

[Positive Case: 54](#_Toc480913707)

# Part I: One Dimensional Finite Element Solution

## 1-D Problem:



For  with boundary conditions  and . The true solution for both the positive and negative cases follows the same form, where there is both a homogenous and negative solution.



The homogenous solution for both cases is derived as follows:















For the positive case:





For the negative case:





For both cases, since the right hand side is linearly related to x, we express the particular solution as:



For the positive case our solution is now:



From our boundary conditions:

,

,

Thus, for the positive case our exact solution is:



Following the same process for the negative case:

,

,

The exact solution for the negative case is now:



## Approximated Solution (Compact 2nd Order):

Consider finding the values of and  using Taylor series, and a known value of  where ,  and .





Adding the two approximations yields:



Rearranged, and ignoring terms higher than the 2nd order, the second derivative is expressed as:



Inputting this in for the negative case yields:



Rearranging makes the **negative** case:



Similarly, the **positive** case is:



## Approximated Solution (Compact 4th Order):

Consider the sum and  using Taylor series. This can be rearranged to include to include the fourth order derivative in the expression for the second order derivative.



The fourth order derivative of U can be represented in terms of the second derivative by diffientiating the original equation twice.





Examining the positive case:



For the sake of simplicity:

Let 

 becomes:



The initial differential equation becomes:



Which simplifies the positive case to:



For the Negative case:



Gamma is expressed as:



Thus, our second derivative term is:



When plugged into the initial negative case:



Multiplying through yields:



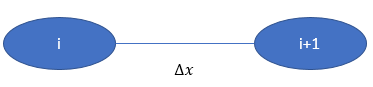
For the final equation in both cases, a matrix was formed to simultaneously solve for every value of U, at x values apart from one another.

## Modifying for Assembly:

The equations derived will be modified to fit an elemental form, that will then be assembled into a global matrix for the left-hand side and right-hand side. All U values will then be solved simultaneously.

### Element:

The elements consist of two nodes, of which are numbered from left to right. The element length is equivalent to . A generic element is given below.



### Ghost Value:

It is obvious when comparing the equations to the elements, there is one node missing to solve the equation at either node within the element. This means an expression must be made for this “ghost term”, in this case, .



The 1st order derivative solved by Taylor series is :



This is rearranged to yield the following equations to define the ghost values.





Since all solutions follow the following patter:



Where and  were previously defined for each case, the solution can be written in either of the following forms:



This gives us the following elemental equations:

2nd Order Negative:



4th Order Negative:



2nd Order Positive:



4th Order Positive:



\*It should be noted that the derivatives left in the element equations serve as jump conditions. Because there is no point loads in the problem, these terms were neglected.

## Post-Processing:

To verify the Solution, an approximated value of the first order derivative evaluated at x=1 was compared with the exact solution. Since the step size is limited to the available computing power, Richardson Extrapolation is a good tool for verifying the convergence of the solution.

### 1st Derivative:

The derivative of the positive case is:



The 2nd order approximation for this case is:



The 4th order approximation for this case is:



The derivative for the negative case is:



The 2nd order approximation is:



The 4th order approximation is:



### Error:

The derivatives for both cases were evaluated at x=1 for the real and approximated solutions. For both cases the error is:



### Richardson Extrapolation:

Richardson Extrapolation is used to extrapolate points ahead based on known values. In the 1-D case, the extrapolation is of . As the number of nodes increases, the magnitude of  changes, thus the values being interpolated are  for each mesh. This is **not** interpolating the derivative at the point past x=1. The following is the equation for the extrapolated value where ***I*** is the iteration which corresponds to a specific mesh size.

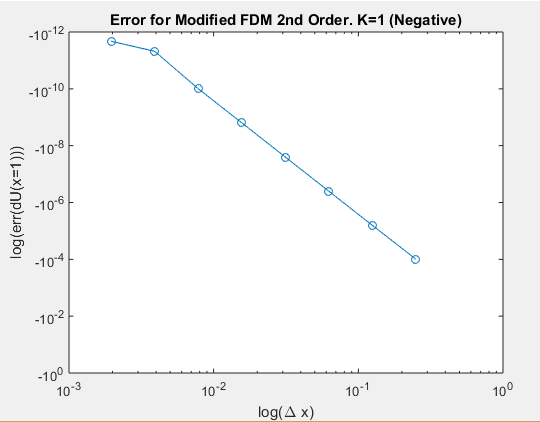
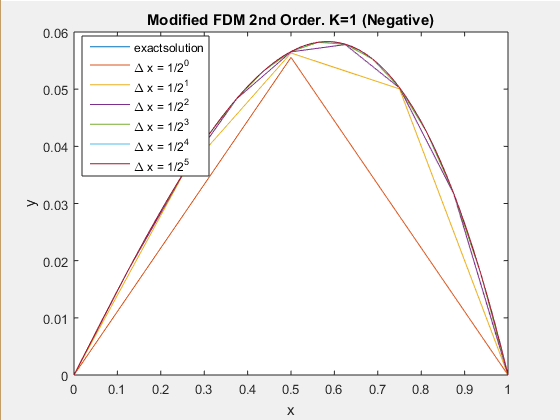


The rate of convergence is how the solution is verified. For the 2nd order solution, =2, and for the 4th order solution, =4. These may not be obtained immediately, and as step size increases, round of error may ensue.

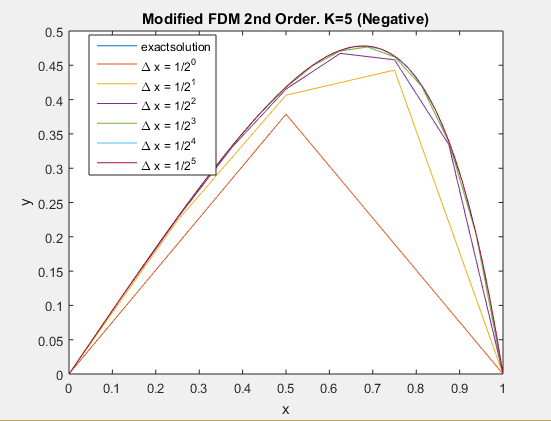


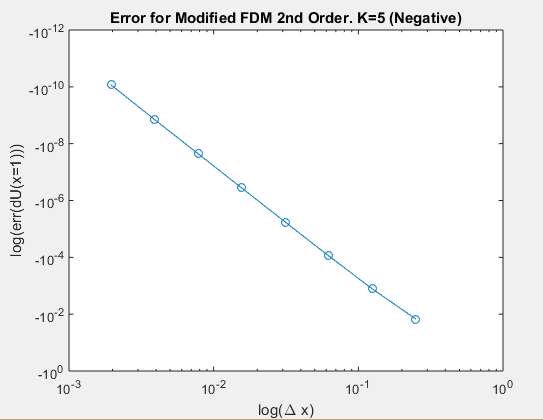
## Results:

### Negative Case:

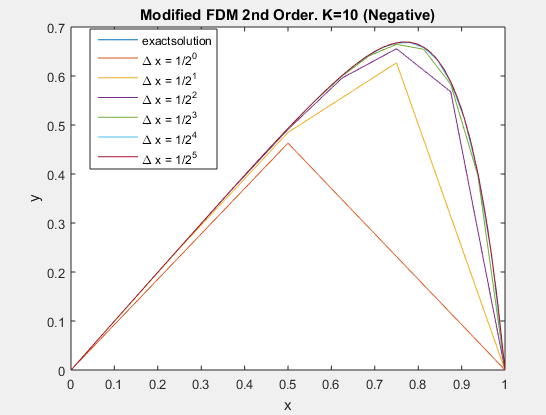


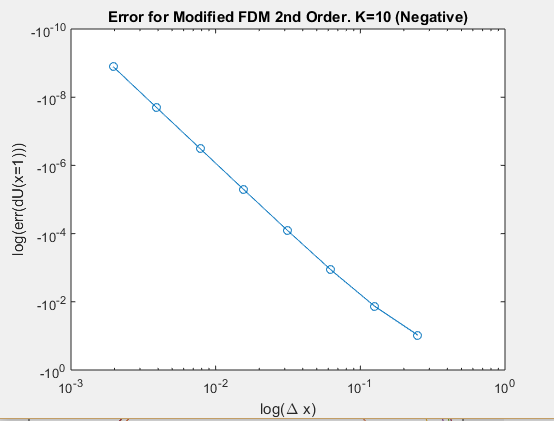
|  |  |
| --- | --- |
| β | 1.98613259693354  1.99644848260351 1.99910664599766 1.99977631648962 1.99994405833558 1.99998599185276 1.99999666381570 1.99999840550580 |



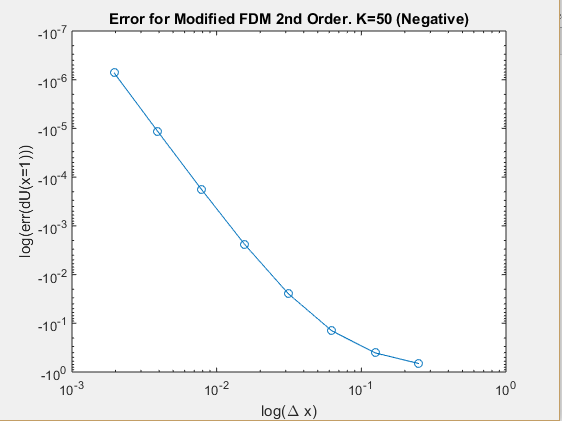
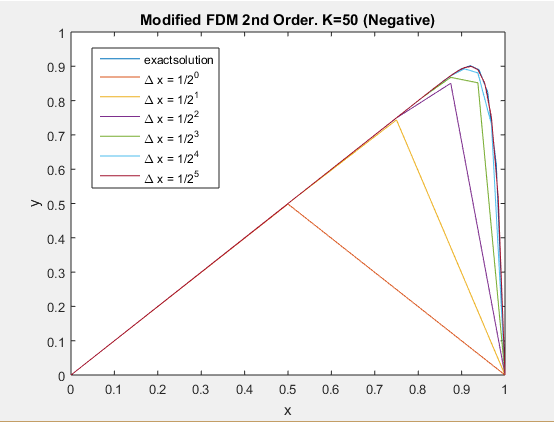


|  |  |
| --- | --- |
| β | 1.68100157647480 1.88766163280889 1.96843440680897 1.99184786366774 1.99794483047400 1.99948512409020 1.99987120854880 1.99996802275551 |

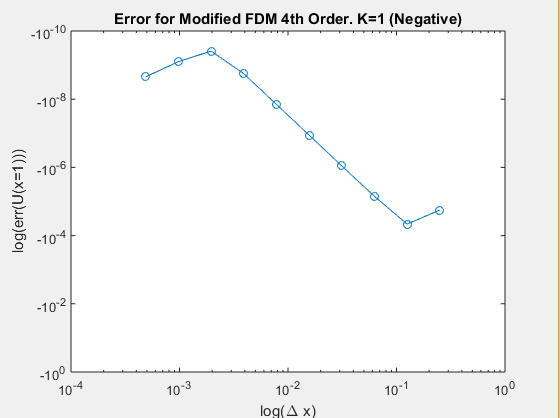
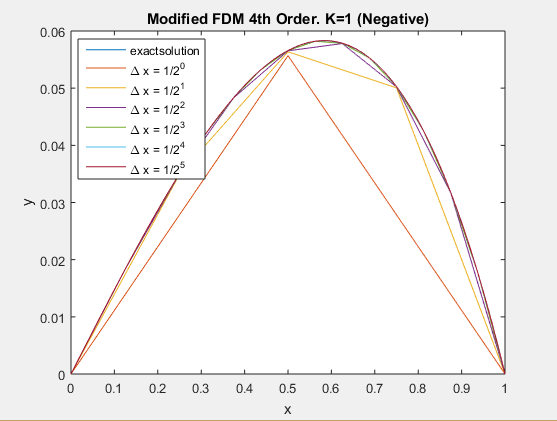




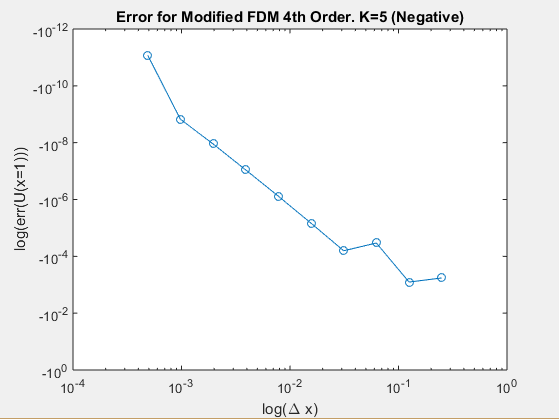
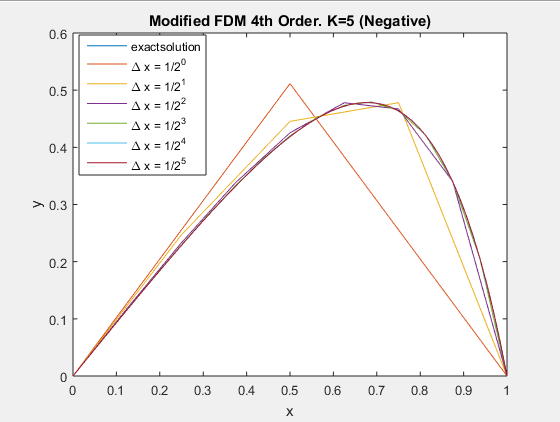
|  |  |
| --- | --- |
| β | 1.88745959697456 1.96838779389783 1.99183645831762 1.99794199458647 1.99948441294808 1.99987102888270 |



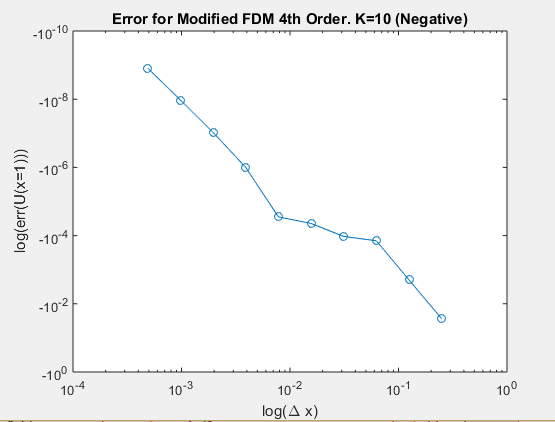
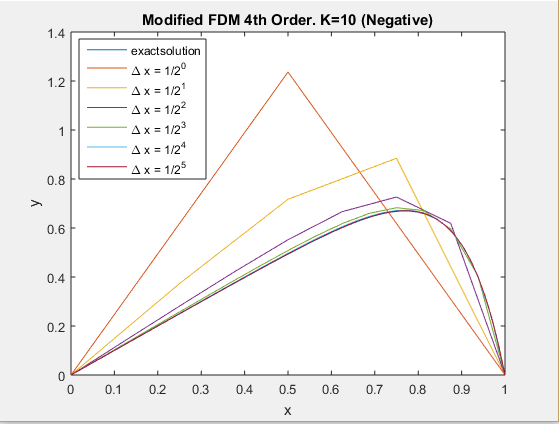
|  |  |
| --- | --- |
| β | 1.02664685979835 1.09607664180835 1.28273770135695 1.58465981558880 1.83711481819385 1.95174641505952 1.98732328407131 1.99678942370786 |



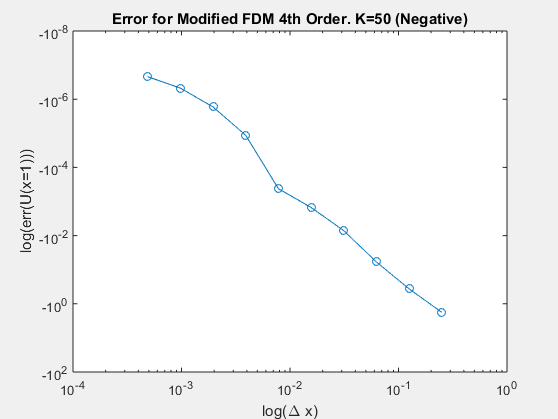
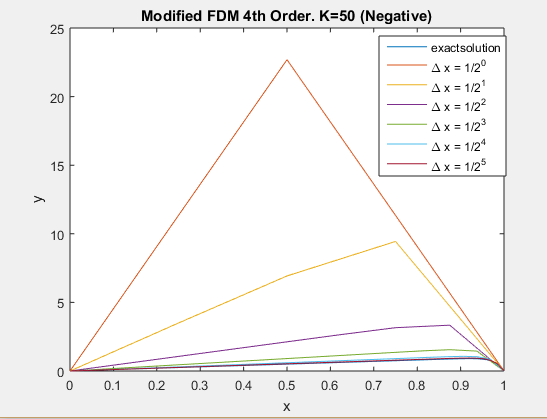
|  |  |
| --- | --- |
| β | 1.81676267923820 1.90580535195356 1.95254391543385 1.97621050419675 1.98808175977696 1.99340630455053 |



|  |  |
| --- | --- |
| β | 4.05043610175361 4.22359246905691 1.50703288694039 1.25863319490841 1.63706129133448 1.82611711659900 1.91521241265524 1.95816035133234 |

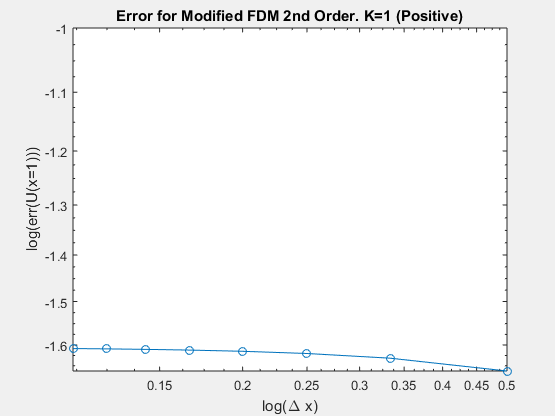
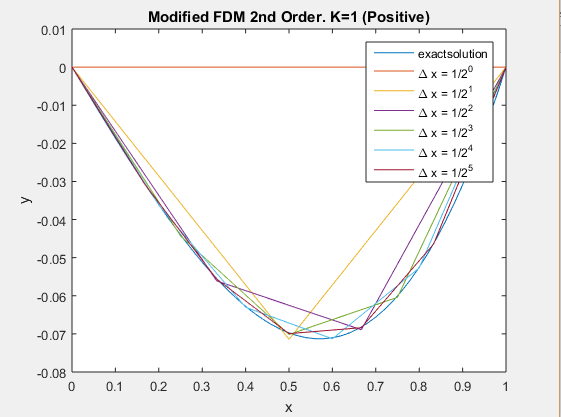


|  |  |
| --- | --- |
| β | 3.44203096317830 4.55230443075600 4.17775922994253 2.70008660982412 2.02506375559067 0.69756790396447 1.57286007248585 |

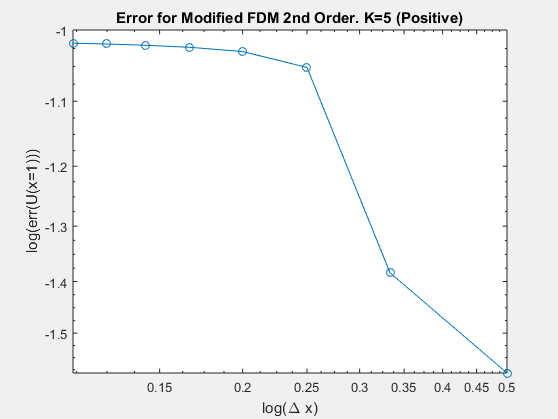
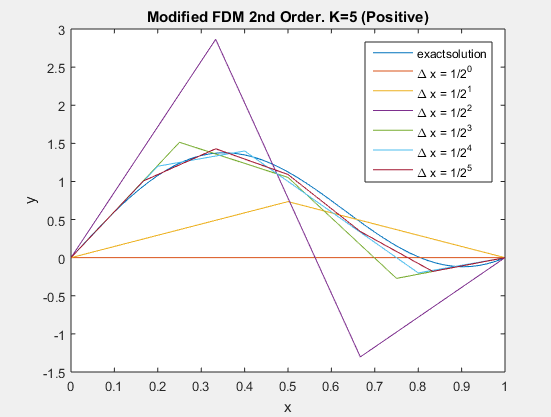


|  |  |
| --- | --- |
| β | 2.92125854325581 3.06519859103628 3.38144534635167 4.30215391657520 3.98593467977111 1.50052520923511 2.70619125002497 3.17995934329542 |

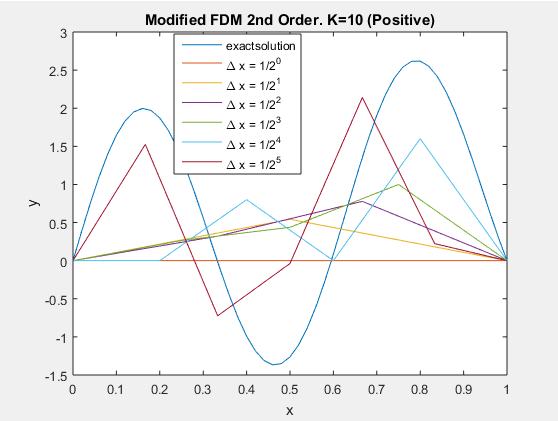
### Positive Case:



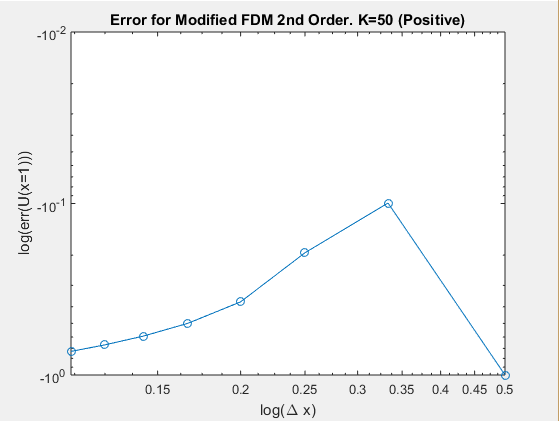
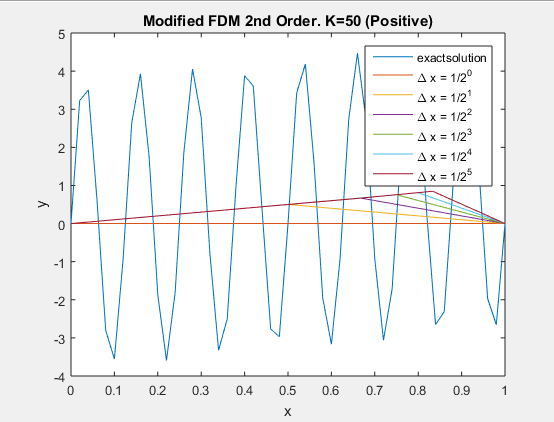
|  |  |
| --- | --- |
| β | 2.46097378344571 1.52001587082605 1.11339073240958 0.881402921455069 0.730399249705638 0.623966925869049 0.544795066811893 0.483549869443175 |



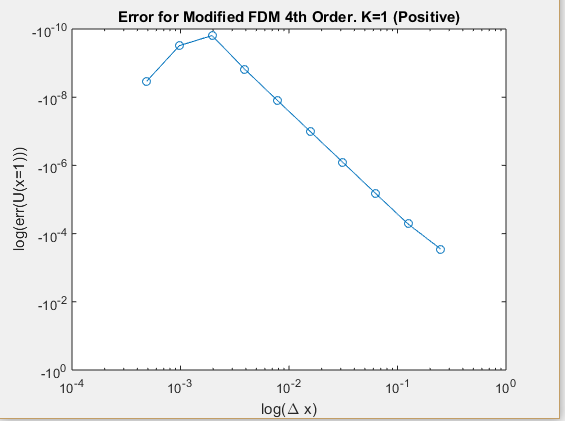
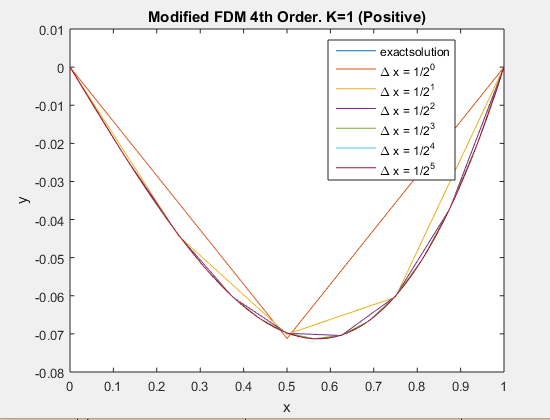
|  |  |
| --- | --- |
| β | 1.22908707391046  -0.229085307485336 2.43748939664495 1.09848724834897 0.809956752646388 0.662680478400809 0.566809140356789  0.497377144999742 |



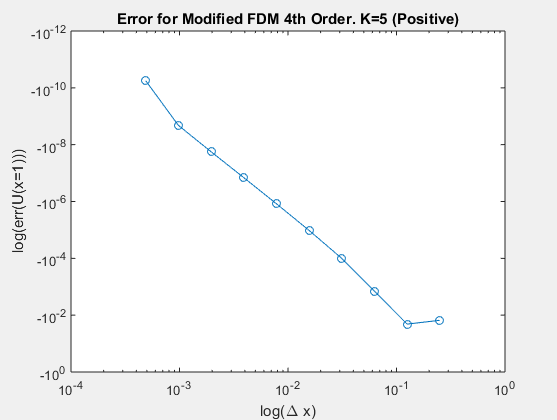
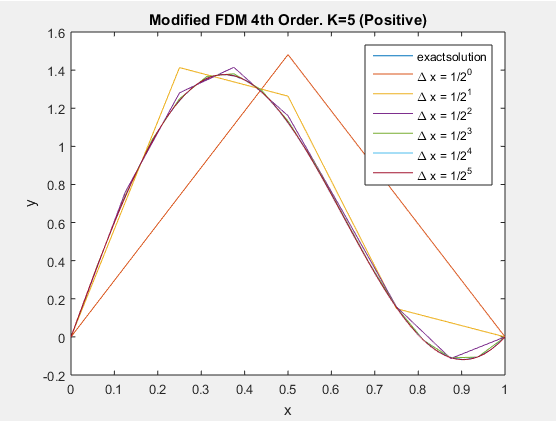
|  |  |
| --- | --- |
| β | 1.22908707391046  -0.229085307485336 2.43748939664495 1.09848724834897 0.809956752646388 0.662680478400809 0.566809140356789  0.497377144999742 |



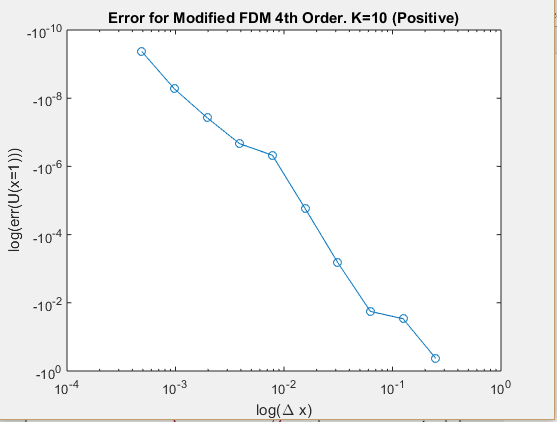
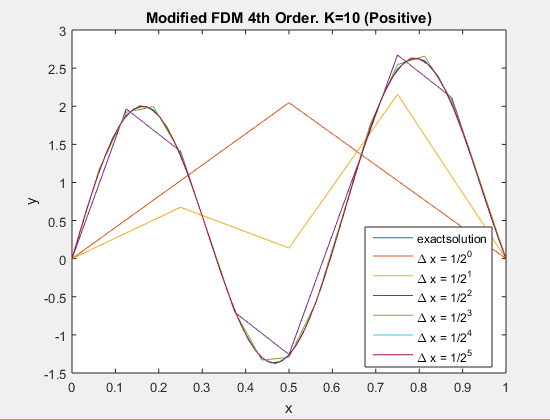
|  |  |
| --- | --- |
| β | 2.30788388068909 1.56602502278845 1.19836892521428 0.970636128878068 0.812967779226840 0.695862805083376 0.604390627792327 |



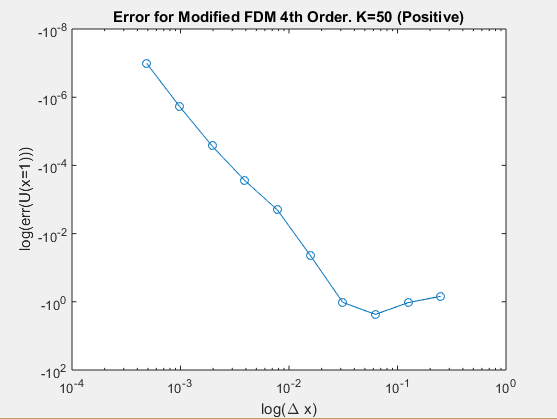
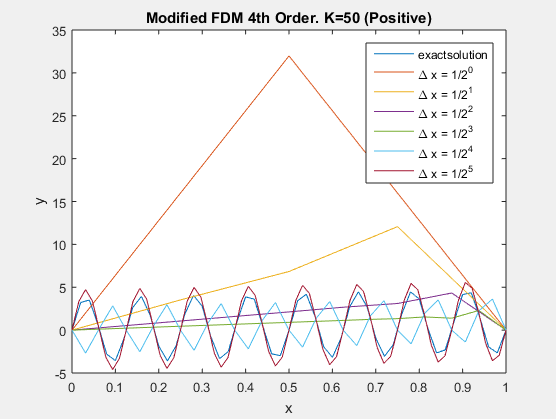
|  |  |
| --- | --- |
| β | 3.29657374001967 3.60210311907131 3.79774242148116 3.89874082674853 3.94940243736878 3.97475484972450 3.98751788721723 3.98874855100204 |



|  |  |
| --- | --- |
| β | 3.03437518326481 1.47110168617570 3.04190010504502 3.58575454550596 3.80612393727135 3.90604758941133 3.95373106437072 3.97710702439277 |



|  |  |
| --- | --- |
| β | 7.58592414715571 6.57619662950185  -0.594414996490762  3.76457983388342  4.00883490506623  4.03302991787432 4.02379106076091 4.01376609726473 |



|  |  |
| --- | --- |
| β | 2.29718776786677  -2.20796847807223  6.30387816223325  7.11626037897871  5.28150545237330  4.39823535578333  4.09705230949028  4.01917948809909 |

# Part II: 2-Dimensional Finite Element Solutions

## Problem:

,



 and  or , 

and  or , 

### Exact Solution for Negative Case:

For partial differential equations, we cannot directly derive an analytical solution directly from the equation. Instead we will assume the solution takes the form of a sine series as follows:



The more iterations of this summation, the more accurate this result is. From this solution, we must find the form of . To do this, we will take the weak formulation of the differential equation, where R is the residual.



The residual can then be expressed:



One characteristic of the residual is as follows:



Where:



The integral can then be re-expressed.



Given that , the equation can be rearranged to give an expression for .



The double integral leftover must be solved as follows:



Consider the following derivative found using chain rule:



We can rearrange this to solve for the x-component of the double integral.



Since sine is equivalent to zero at x=0 and x=1, the first component is zero. The second is given by the fundamental theorem of calculus.



Thus:



The total derivative is then:



This makes completes our terms for  and U(x,y).





### Exact Solution for Positive Case:

The exact solution for the positive case is very similar to the negative case, except for a sign change in a portion of the denominator.  




### 2nd Order Approximation:

Consider the taylor series expansions of  and about , or rather  and  about .



Subtracting the two, and rearranging gives a second order approximation of .

Similarly, consider the expansions of  and about . This gives the second order approximation of .



Inserting the approximations into the initial differential equation yields:



If we assume that the mesh is square(), then:



The solution for the **negative** case is:



The solution for the **positive** case:



### Fourth Order Solution(Negative):

Recall the following terms to derive the second x and y derivatives:





These can be rearranged to give us a better approximation in terms of the 4th derivatives.



Where:



Note that . We can derive this as follows.









This makes the second derivatives with respect to x and y:





Placing these terms into the initial differential equation and multiplying through by  yields:





### Fourth Order Solution(Positive)

The positive case will follow the same form as the negative case, but now the fourth derivatives are:



Giving us the following fourth order approximations for the second derivatives.





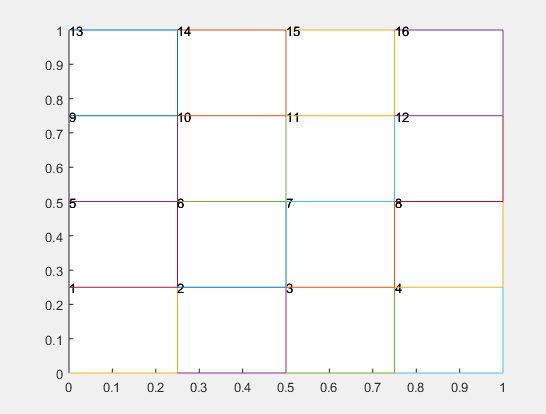
The solution is now:



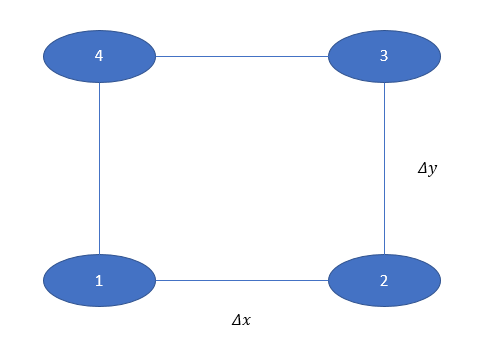


### Mesh and Elements:

Before solution can be modified to fit into elements, the mesh needs to be stated, as well as the nodal numbering system in the global & elemental frame. In the global frame, the elements and the nodes are numbered from left to right until they reach the edge, which then they start at the far left node from the next row up. This is shown in an example mesh below consisting of 16 elements.

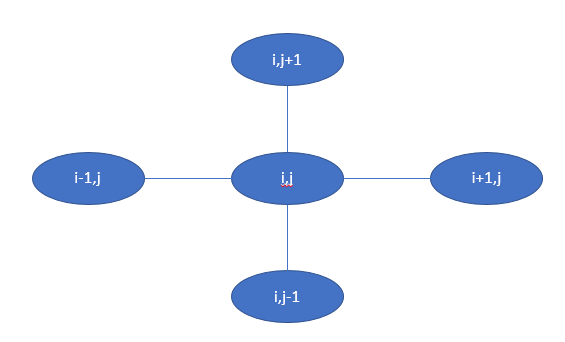


In the elemental frame, nodes are numbered from the bottom left node counter-clockwise, as shown below.

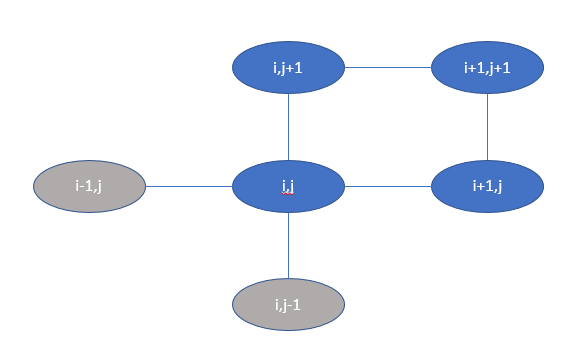


### Ghost Points(2nd Order)

In two dimensions, both second order solutions require 5 points as seen below.



Unfortunately, the individual element does not contain  or , so they must be expressed differently, These points are the ghost points.



It is assumed that the derivative with respect to the x and y directions is known, so that an expression can be made for the ghost points.

From the derivative in the x-direction:



From the derivative in the y-direction:



Like the 1-d problem, the solutions for positive and negative follow the same scheme. In this case it is:



Rearranging this equation with the expressions for the ghost points yields:



The **negative** case is then:

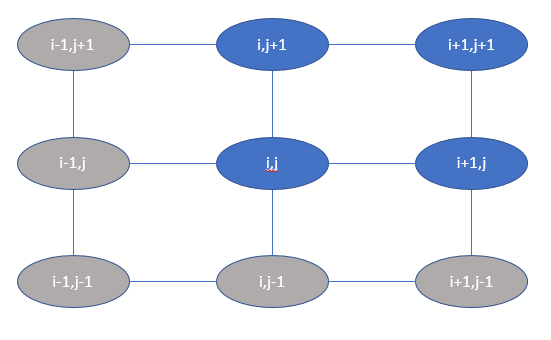


The **Positive** case is:



### Ghost Points(4th Order)

Both fourth order solutions require 9 points for a single equation. This means that 5 nodal values will be represented as ghost points within each element. Terms for  and  have already been established, however terms must be written for , , and .



 comes from the y derivative of 



 is given by the x derivative of.



 is defined by the x derivative of , which was defined previously.





The fourth order solutions both follow the following scheme:



It should be remembered that the derivatives on the right hand side actually function as the jump condition. Because there are no point loads, these terms will go to zero upon assembly.

The **Negative** element solution is:



The **Positive** element solution is:



## Post-Processing:

Rather than approximate a derivative to verify the solution, the actually value of U(0.5,0.5) was compared with the solution values at that point.

### Error:

The error was based on the difference in midpoint values.



### Richardson Extrapolation:

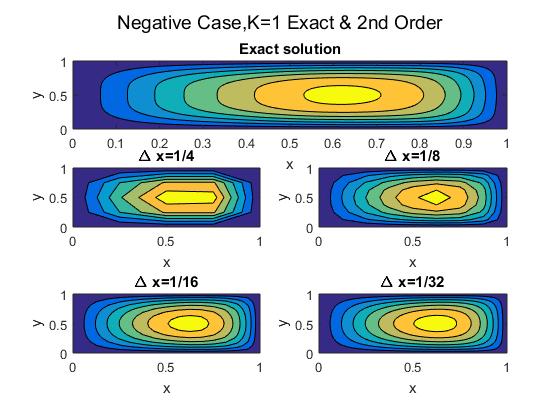
Richardson Extrapolation and rate of convergence equation for the midpoint value was identical to that used for the 1-dimensional problem.

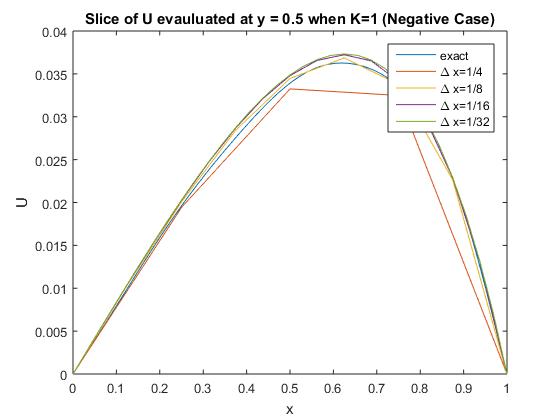


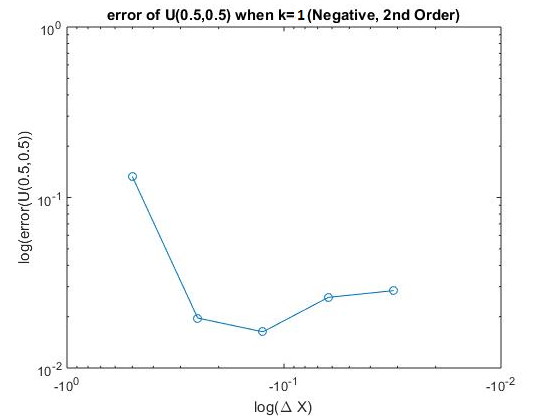


## Results:

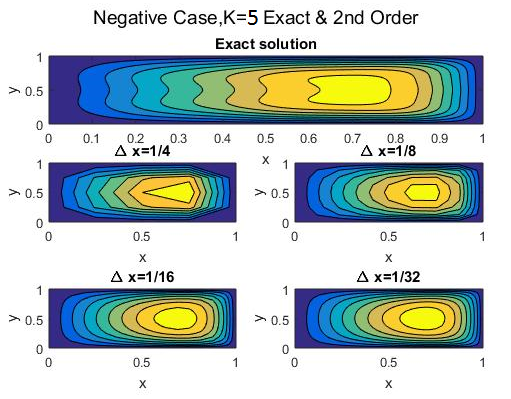
### Negative Case:

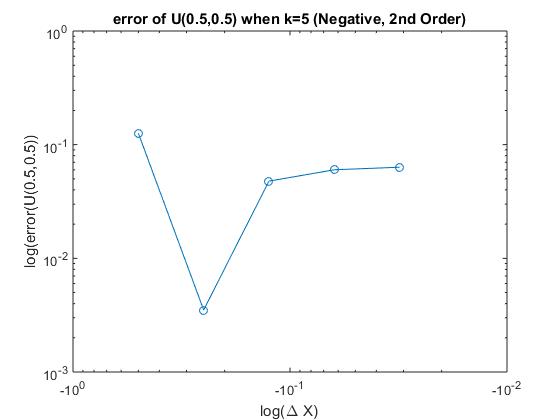
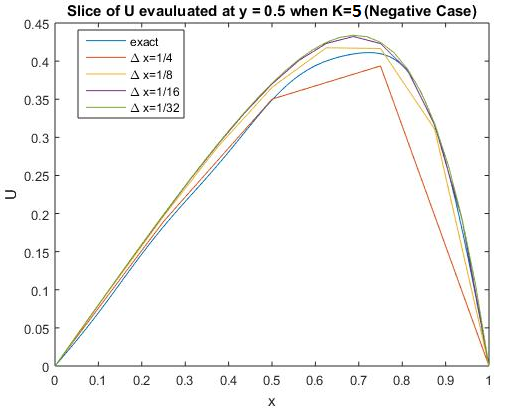




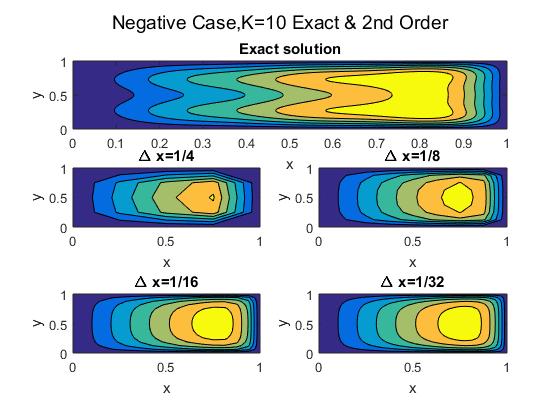


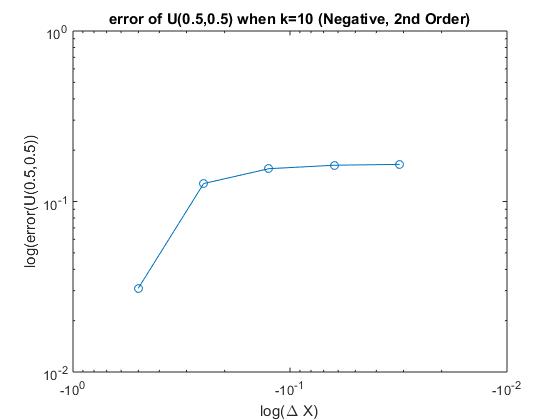
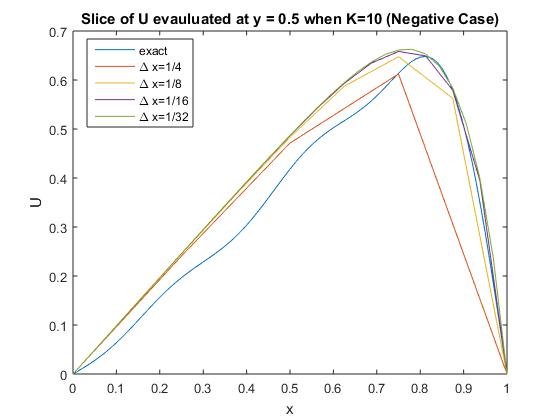
|  |  |
| --- | --- |
| β | 1.65370701525297 1.89420127992649 1.97166329288864 |



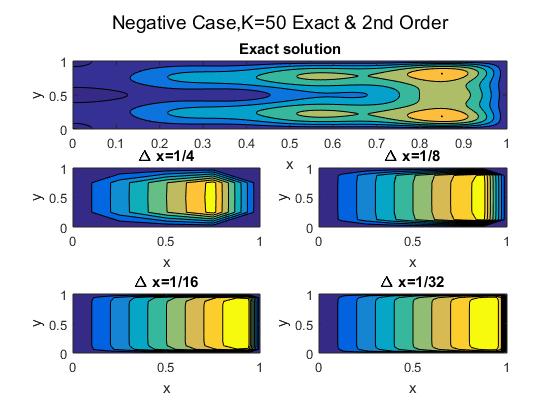


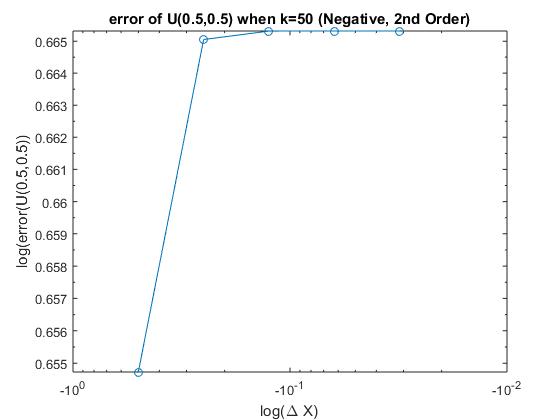
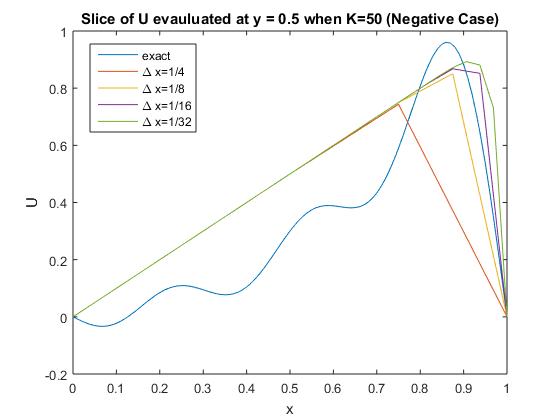
|  |  |
| --- | --- |
| β | 1.65370701525297 1.89420127992649 1.97166329288864 |



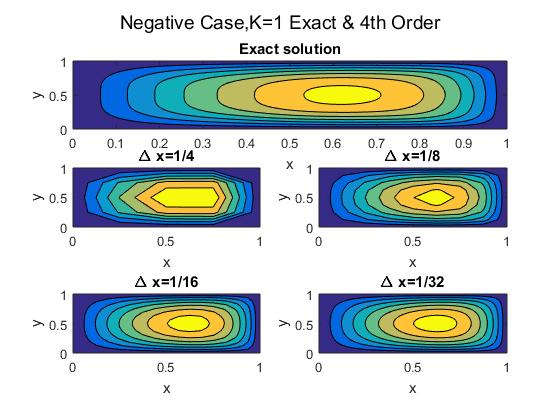


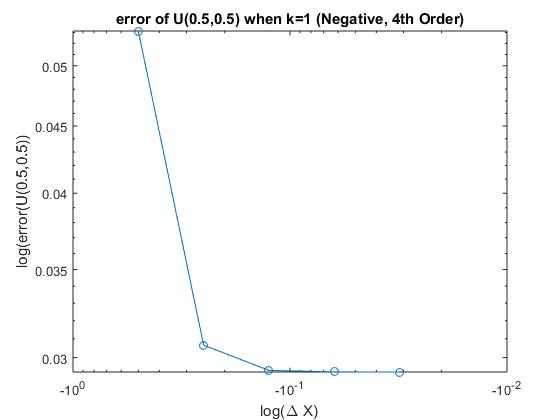
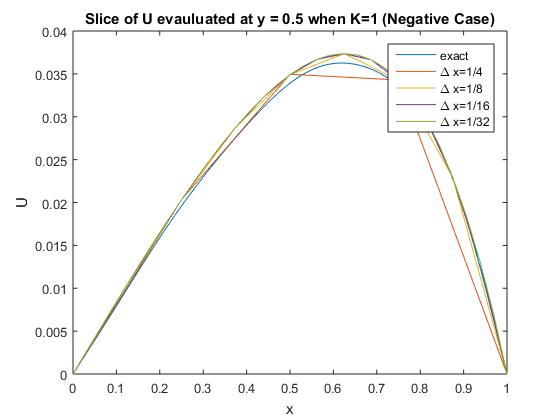
|  |  |
| --- | --- |
| β | 1.76838167659754 1.92793154620969 1.97839840335562 |



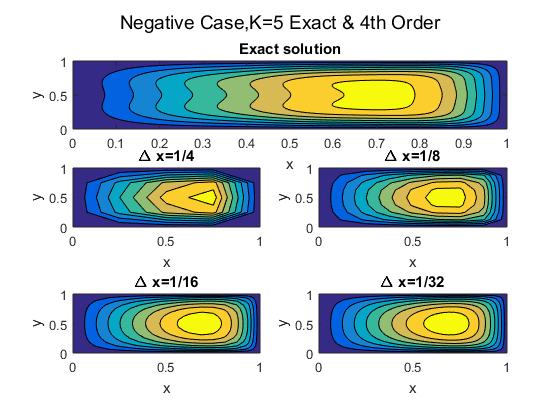


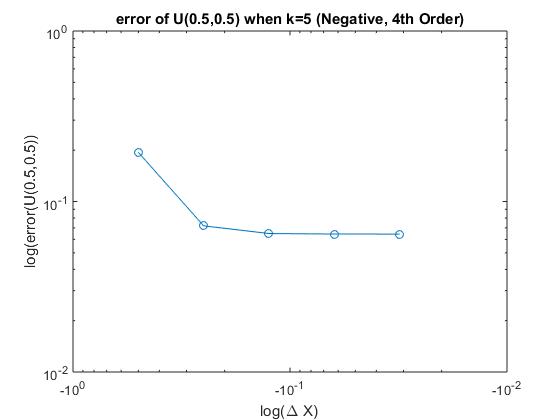
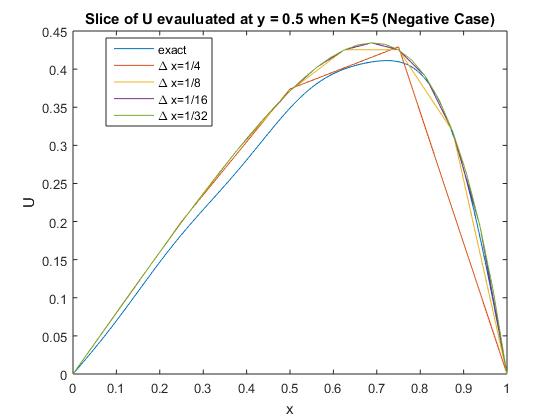
|  |  |
| --- | --- |
| β | 5.29152998372136 6.82232272196261 6.97238286859709 |



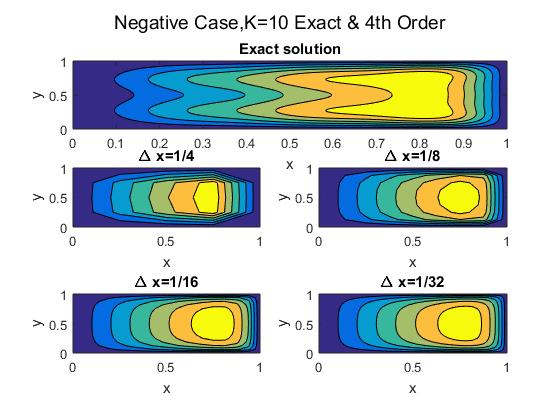


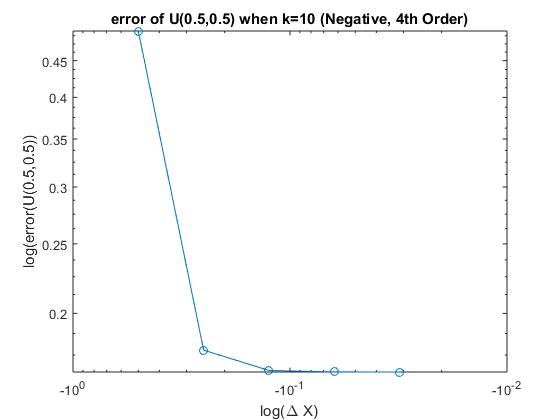
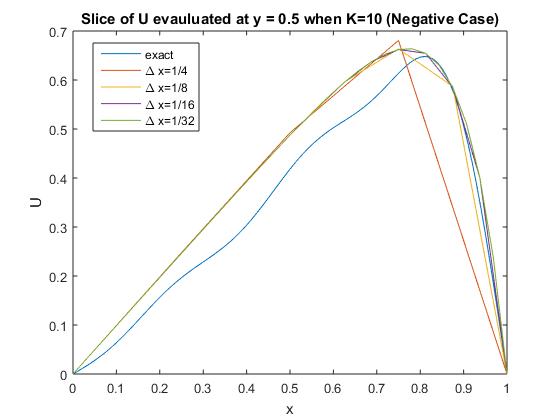
|  |  |
| --- | --- |
| β | 4.09080876151641 4.05573087755940 4.01426280988517 |



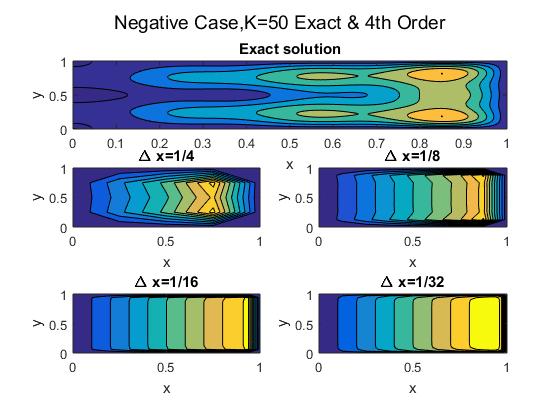


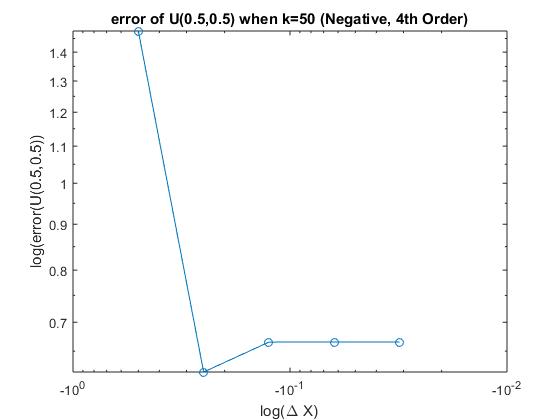
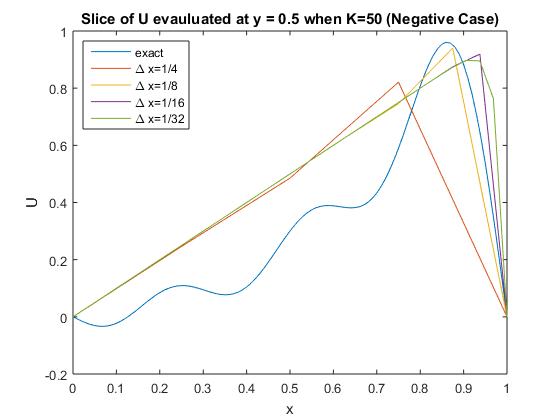
|  |  |
| --- | --- |
| β | 4.10791607853529 3.97709191533730 3.98791977770055 |





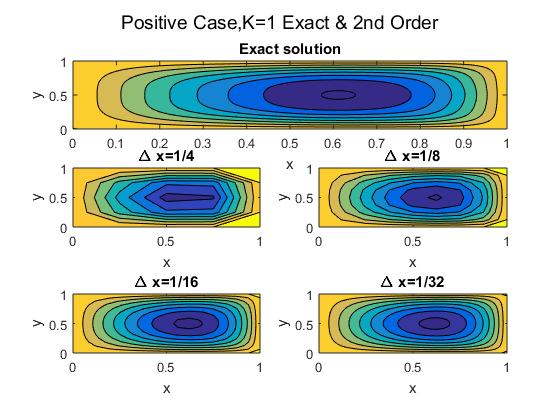
|  |  |
| --- | --- |
| β | 4.82099203017053 3.87810438508257 3.94215666309746 |

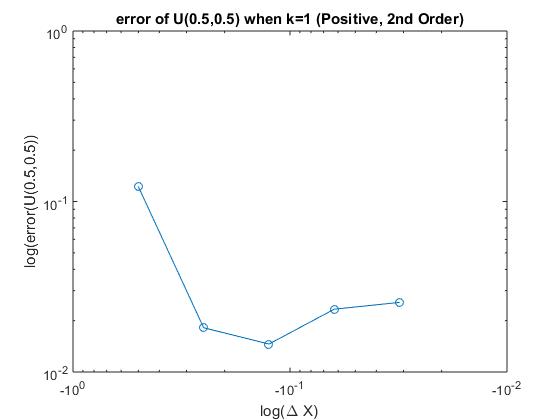
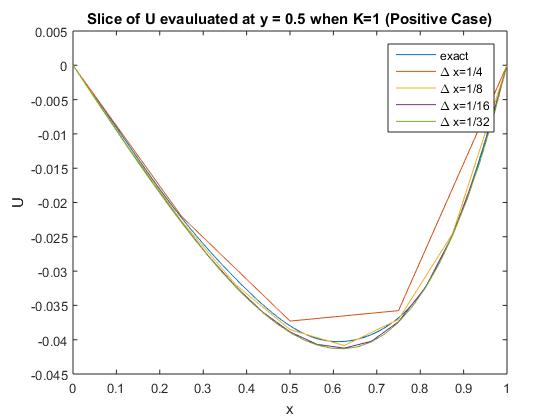




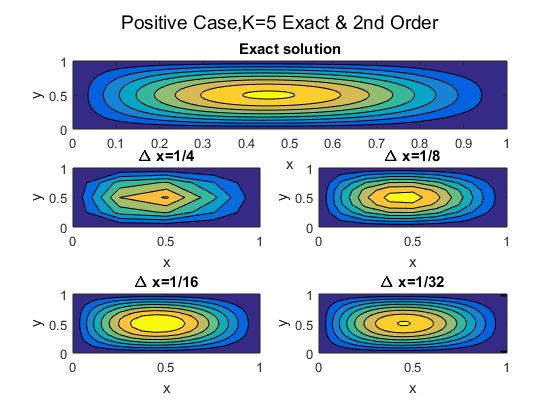
|  |  |
| --- | --- |
| β | 4.13156415 + 4.5323i 8.70386847815392  20.785898 + 4.53236i |

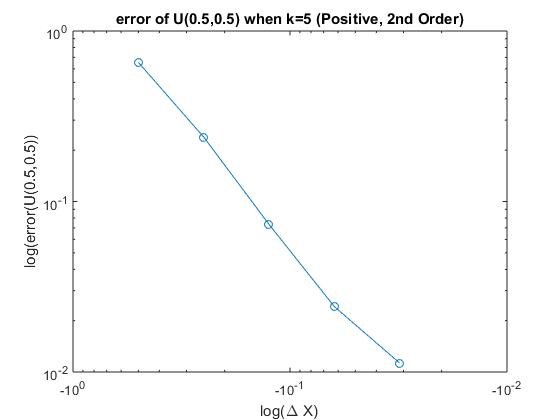
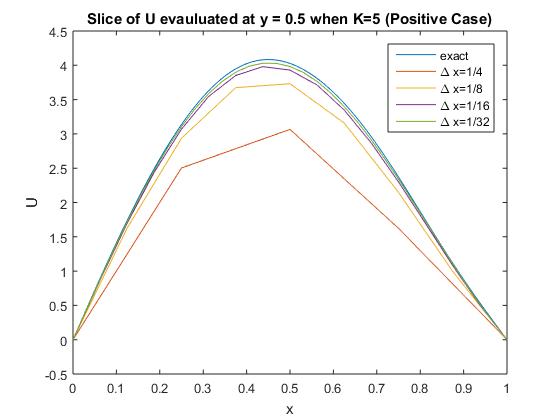
### Positive Case:



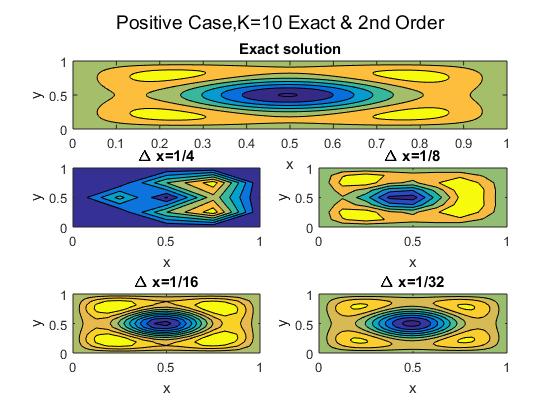


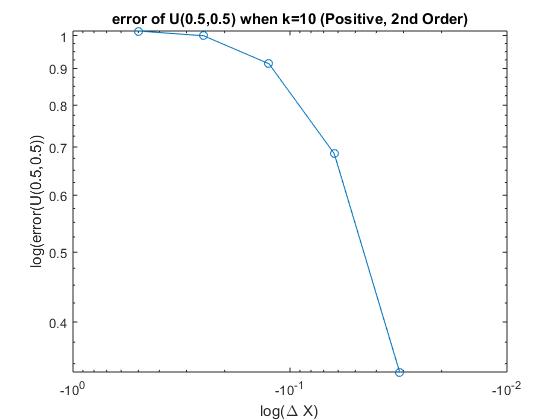
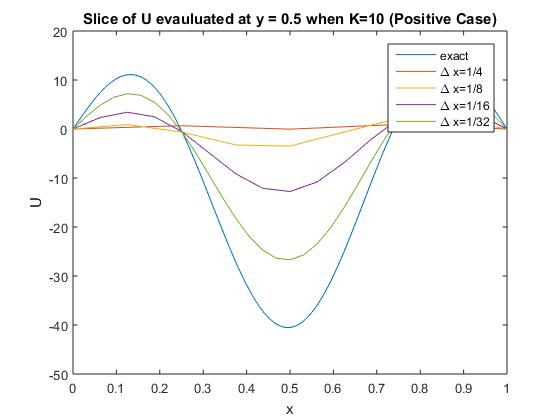
|  |  |
| --- | --- |
| β | 1.66764379182005  1.89991307159311  1.97332538361027 |



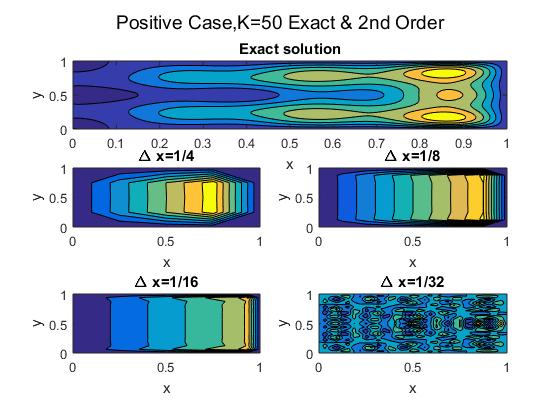


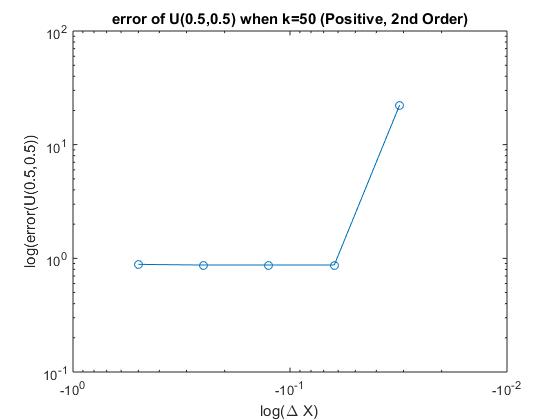
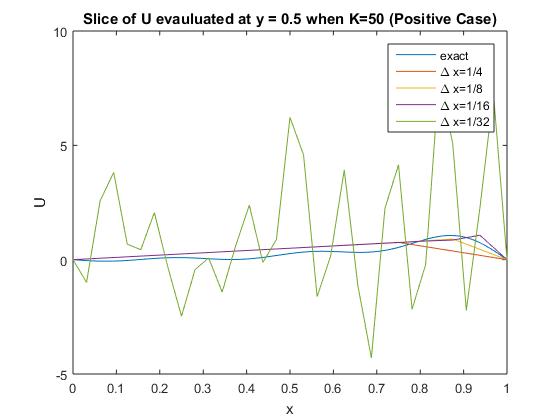
|  |  |
| --- | --- |
| β | 1.33268923507226  1.74712383992894  1.93118143025936 |



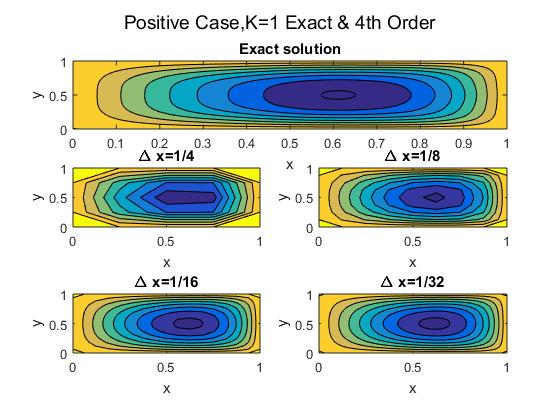


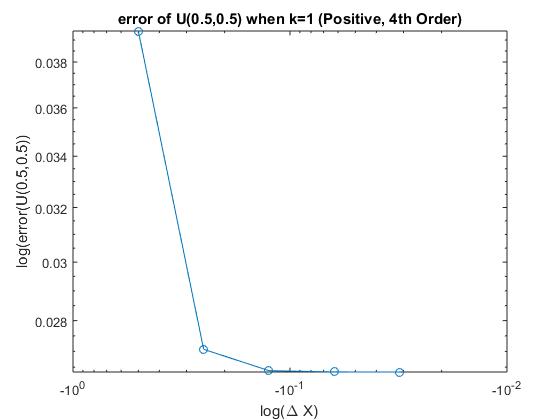
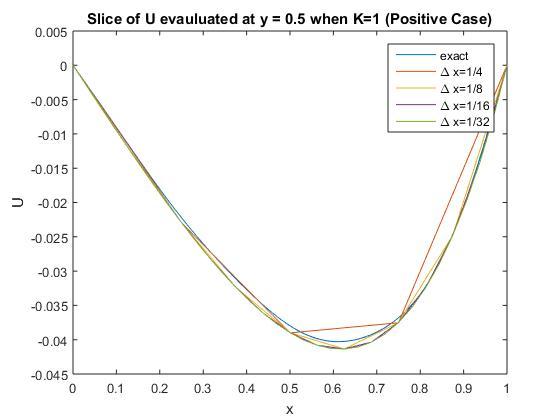
|  |  |
| --- | --- |
| β | -2.47232911195255  -1.41697842521752  -0.591322154862871 |



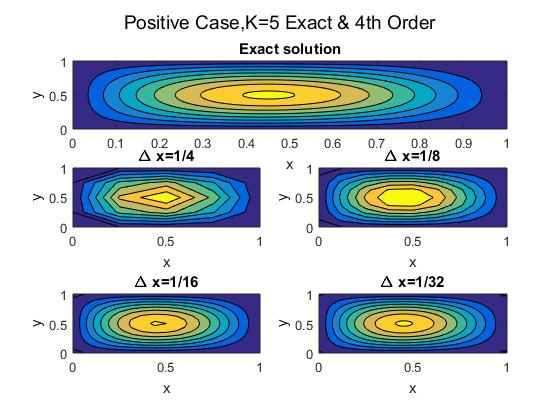


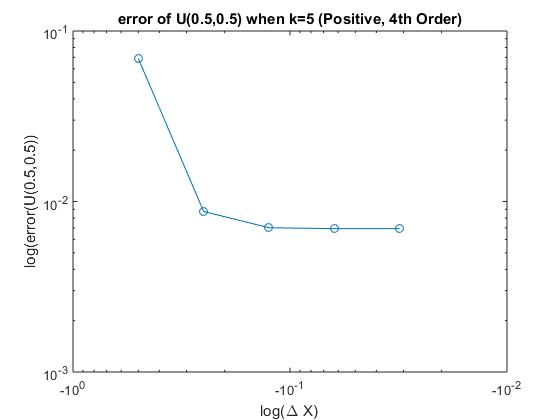
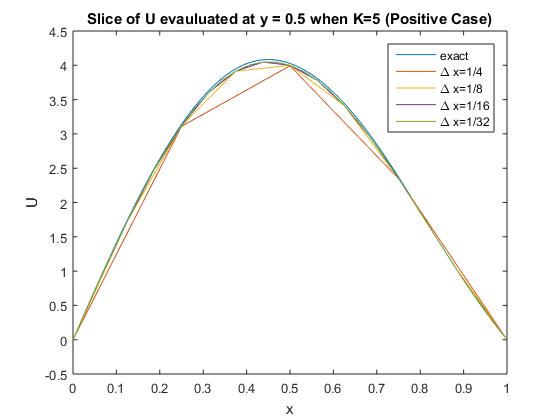
|  |  |
| --- | --- |
| β | 0.31558 + 4.5323601 6.42264757650162 22.4964274198006 |



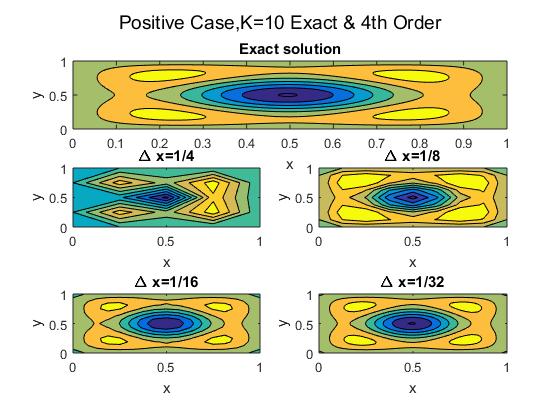


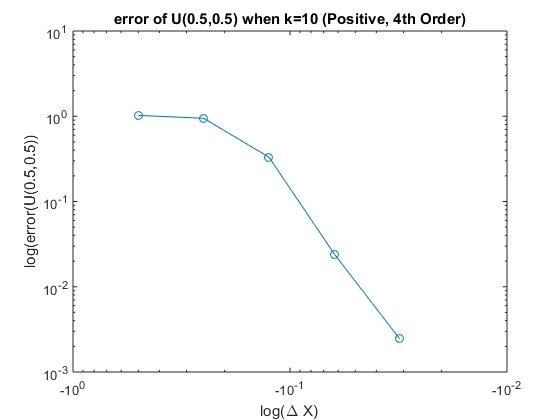
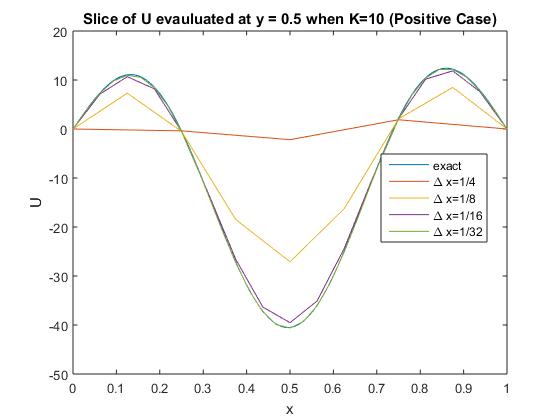
|  |  |
| --- | --- |
| β | 4.1929032183028  4.11975814750246 4.03212425879795 |



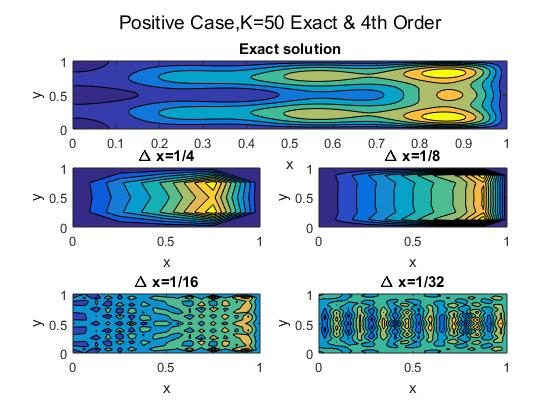


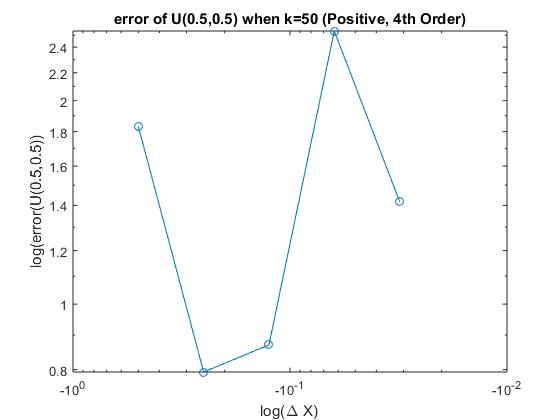
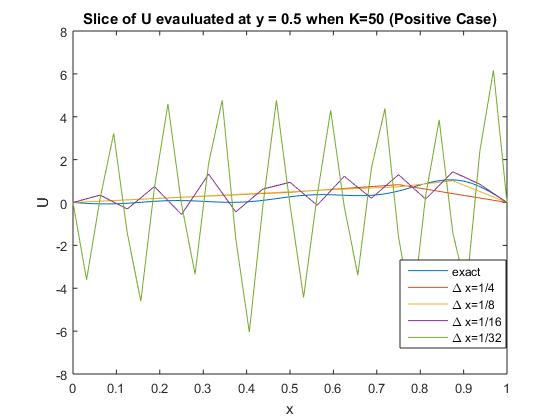
|  |  |
| --- | --- |
| β | 5.12685761431336 4.39959833313171 4.11204060531191 |





|  |  |
| --- | --- |
| β | -2.99886646213561 1.00360603620706 3.52409205953544 |





|  |  |
| --- | --- |
| β | 3.75830 + 4.5323i  -4.439703  -1.2473675+ 4.53239i |

# Appendix:

## 2-D Code:

The following code is for a 2nd order positive case. The same form applies for all 1-D cases.

%% 1-d 2nd Order Positive

clc; clear all; close all;

xi = 0; xf = 1; k =5; i = 13;

elements = i; Nodes = elements + 1;

%u value and dU(1)

x = xi:1/50:xf;

u = x-sin(k\*x)/sin(k);

dUp\_true = 1-k\*cos(k\*x)/sin(k);

j = 1;

ek = zeros(2,2)

for elements = 1:i

N = elements

gk = zeros(N+1); gf = zeros(N+1,1);

del\_x(elements) = (xf - xi)/N;

dx = del\_x(elements);

X = linspace(xi,xf,N+1);

%Build elemental matrices

for i = 1:N

%element stiffness

ek(1,1) = -2 + k^2\*dx^2;

ek(1,2) = 2;

ek(2,1) = 2;

ek(2,2) = -2 + k^2\*dx^2;

%element load

ef = [k^2\*X(i)\*dx^2;k^2\*X(i+1)\*dx^2];

%global stiffness

gk(i,i) = gk(i,i)+ ek(1,1);

gk(i+1,i) = gk(i+1,i) + ek(2,1);

gk(i,i+1) = gk(i,i+1) + ek(1,2);

gk(i+1,i+1) = gk(i+1,i+1) + ek(2,2);

gk

%global loads

gf(i) = gf(i) + ef(1);

gf(i+1) = gf(i+1) + ef(2);

gf

%refifine x values

end

%apply bc via penalty method

gk(1,1) = gk(1,1)+1e20; gk(N+1,N+1) = gk(N+1,N+1)+1e20;

%SOLVE FOR DEFLECTION

U = gk\gf;

if j <2

U1 = U;

X1 = X

GK1 = gk

elseif j <3

U2 = U;

X2 = X;

GK2 = gk

elseif j < 4

U3 = U;

X3 = X;

GK3 = gk

elseif j < 5

U4 = U;

X4 = X;

elseif j < 6

U5 = U;

X5 = X;

elseif j < 7

U6 = U;

X6 = X;

else

end

dU(j) = (U(N+1)-U(N))/del\_x(j);

dU2(j) = (U(N+1)-U(N))/del\_x(j)+(del\_x(j)/2)\*k^2;

err(j) = abs((dUp\_true-dU2(j))/dUp\_true\*100);

if j > 4

RE\_dU = (dU2(j-1)^2-dU2(j-2)\*dU2(j))/(2\*dU2(j-1)-dU2(j-2)-dU2(j));

err\_RE(j) = abs(RE\_dU - dU2(j-2))/abs(RE\_dU)\*100;

B(j) = (1/log10(2))\*log10((RE\_dU-dU2(j-2))/(RE\_dU-dU2(j)));

end

j = j+1

end

B

plot(x,u,X1,U1,X2,U2,X3,U3,X4,U4,X5,U5,X6,U6)

xlabel('x'); ylabel('y')

title('Modified FDM 2nd Order. K=5 (Positive)')

legend('exactsolution','\Delta x = 1/2^0','\Delta x = 1/2^1','\Delta x = 1/2^2','\Delta x = 1/2^3','\Delta x = 1/2^4','\Delta x = 1/2^5','\Delta x = 1/2^6')

figure

loglog(del\_x,-err\_RE)

title('Error for Modified FDM 2nd Order. K=5 (Positive)')

ylabel('log(err(U(x=1)))'); xlabel('log(\Delta x)')

## 4-D Code:

## Main code:

clc; clear all; close all;

k = 50;

%exact Solution

[Xt,Yt,Ut,dUt,midt,xx, slice\_exact] = exact(k,1);

J = 1;

%define cartesian shape(in this case, a 1x1 square)

xi = 0; xf = 1; yi = 0; yf = 1; iterations = 5;

%number of segments in x&y direction

for j = 1:iterations

Nx = 2^j; Ny = Nx;

%Generate Mesh, &nodes

[XY,nodes,dx,dy] = mesh\_square(xi,yi,xf,yf,Nx,Ny);

%number of elements and nodes

nelem = length(nodes); nnodes = length(XY);

%define size of global stiffness and load matrices

gk = zeros(nnodes);

gf = zeros(nnodes,1);

for nel = 1:nelem

%build element matricies

%1-negative 2nd order; 2-positive 2nd order; 3-negative 4th order; 4-positive 4th order

[ek,ef,node] = elem2D(XY,nel,k,dx,nodes,1);

%compile matrices

[gk,gf] = assemble2D(ek,ef,gk,gf,node);

end

%Apply BC(All Edges are wquivalent to Zero)

for i = 1:length(XY)

if mod(XY(i,1),1) < 0.0001

gk(i,i) = gk(i,i) + 1e20;

gf(i) = gf(i);

elseif mod(XY(i,2),1) < 0.000001

gk(i,i) = gk(i,i) + 1e20;

gf(i) = gf(i);

else

gk(i,i) = gk(i,i);

gf(i) = gf(i);

end

end

%solve U

lin\_U = gk\gf;

n = Nx;

%U is currently linearized. It must be modified to fit mesh.

Ufull = zeros(n+1,n+1);

for i = 1:n+1

for j = 1:n+1

Ufull(i,j) = lin\_U((i-1)\*(n+1)+j);

end

end

x = []; x(1) = 0;

for i= 1:n

x(i+1) = x(i) + dx;

end

[X,Y] = meshgrid(x);

%1-d view of x at y=0.5

for i=1:n+1

slice(i)=Ufull(1+(n)/2,i);

end

%store values for different mesh sizes

if J < 2

X1 = X; Y1 = Y; U1 = Ufull; slice1 = slice; x1=x;

elseif J < 3

X2 = X; Y2 = Y; U2 = Ufull; slice2 = slice; x2=x;

elseif J < 4

X3 = X; Y3 = Y; U3 = Ufull; slice3 = slice; x3=x;

elseif J < 5

X4 = X; Y4 = Y; U4 = Ufull; slice4 = slice; x4=x;

elseif J < 6

X5 = X; Y5 = Y; U5 = Ufull; slice5 = slice; x5=x;

end

%store midpoint,dx,&error for each iteration

mid(J) = Ufull(1+(n)/2,1+(n)/2);

delx(J) = dx;

err(J) = abs(mid(J) - midt)/abs(midt);

J = J + 1;

end

%RE\_mid - Richardson Extrapolation of the midpoint

for j =2:length(delx)-1

RE\_mid(j) = (mid(j)^2-mid(j-1)\*mid(j+1))/(2\*mid(j)-mid(j-1)-mid(j+1));

end

%err\_RE - error of richardson extrapolated value

for j =2:length(delx)-1

err\_RE(j) = abs(RE\_mid(j) - midt)/abs(midt);

end

%B- rate of convergence

for j =2:length(delx)-1

B(j) = (1/log10(2))\*log10((RE\_mid(j)-mid(j-1))/(RE\_mid(j)-mid(j)));

end

%Plot contours

figure

subplot(3,2,[1,2])

suptitle('Negative Case,K=50 Exact & 2nd Order')

contourf(Xt,Yt,Ut);

xlabel 'x'; ylabel 'y'; title 'Exact solution'

subplot(3,2,3)

contourf(X2,Y2,U2)

xlabel 'x'; ylabel 'y'; title '\Delta x=1/4'

subplot(3,2,4)

contourf(X3,Y3,U3)

xlabel 'x'; ylabel 'y'; title '\Delta x=1/8'

subplot(3,2,5)

contourf(X4,Y4,U4)

xlabel 'x'; ylabel 'y'; title '\Delta x=1/16'

subplot(3,2,6)

contourf(X5,Y5,U5)

xlabel 'x'; ylabel 'y'; title '\Delta x=1/32'

%Plot slices

figure

plot(xx,slice\_exact,x2,slice2,x3,slice3,x4,slice4,x5,slice5)

xlabel 'x'

ylabel 'U'

title 'Slice of U evauluated at y = 0.5 when K=50 (Negative Case)'

legend('exact','\Delta x=1/4','\Delta x=1/8','\Delta x=1/16','\Delta x=1/32')

%plot error of midpoint

figure

loglog(-delx,err,'-o')

title('error of U(0.5,0.5) when k=50 (Negative, 2nd Order)'); xlabel('log(\Delta X)'); ylabel('log(error(U(0.5,0.5))');

## Mesh Code:

unction [XY,nodes,dx,dy] = mesh\_square(xi,yi,xf,yf,Nx,Ny)

% ============================================================

% SETS UP A MESH FOR A PIPE WITH FINS (ONLY 1/16 IS MESHED)

% INPUTS: xi -- initial x position

% xf -- final x position

% yi -- initial position

% yf -- final y position

% Nx -- NUMBER OF INTERVALS in x direction

% Ny-- NUMBER OF INTERVALS in y direction

%

% OUTPUT: XY -- COORDINATES OF ALL NODES

% nodes -- MATRIX CONTAINING GLOBAL NODE NUMBER FOR

% EACH LOCAL NODE NUMBER (1 THRU 4)

% BC -- BOUNDARY CONDITION FLAG FOR EACH ELEMENT FACE

% (1 FOR INNER RADIUS, 2 FOR OUTER RADIUS PLUS FIN)

% IF BC(e,i) IS 0, THEN FACE i OF ELEMENT e IS AN

% INTERNAL FACE

% MAT -- MATERIAL NUMBER FOR EACH ELEMENT

% dX -- x increment size

% dY -- y increment size

% ============================================================

Xint = (xf - xi)/(Nx);

Yint = (yf - yi)/(Ny);

dx = Xint; dy = Yint;

% DETERMINE TOTAL NUMBER OF NODES AND TOTAL NUMBER OF ELEMENTS

% ------------------------------------------------------------

global nelem nnodes

nelem = (Nx)^2;

nnodes = (Nx+1)\*(Ny+1);

XY = zeros(nnodes,2);

nodes = zeros(4,nelem);

npt = 1;

for ny=0:Ny

for nx=0:Nx

XY(npt,:) = [xi+nx\*Xint,yi+ ny\*Yint];

npt = npt+1;

end

end

for i = 1:nelem

if i < Nx +1

nodes(1,i) = i;

else if mod((i-1)^2,nelem) <0.001

nodes(1,i) = nodes(2,i-1) +1;

else

nodes(1,i) = nodes(2,i-1);

end

end

nodes(2,i) = nodes(1,i) + 1;

nodes(3,i) = nodes(1,i) + Nx + 2;

nodes(4,i) = nodes(3,i) - 1;

end

nodes = nodes';

%confirms mesh

plot\_mesh(XY,nodes)

%Example Element

% face 3

% 4-------3

% | |

% face 4 | | face 2

% | |

% 1-------2

% face 1

## Mesh Plot:

function plot\_mesh(XY,nodes)

% =========================================================================

% PLOTS THE MESH GENERATED IN THE PREVIOUS STEP, ELEMENT BY ELEMENT, SO THAT

% THE USER CAN MAKE SURE EVERYTHING WAS DONE CORRECTLY. THE EDGES OF THE

% ELEMENT ARE PLOTTED USING STRAIGHT LINES, WHICH MAY NOT BE A TRUE

% REPRESENTATION OF THE ACTUAL PHYSICAL MESH, BUT IT IS A GOOD

% APPROXIMATION.

% =========================================================================

global nelem

% ORDER IN WHICH NODAL COORIDINATES ARE PULLED FROM THE OVERALL GLOBAL

% MATRIX, XY

order = [1 2 3 4];

N = 4;

hold on

% PLOT EACH ELEMENT, AND LABEL THE ELEMENT

% COLOR SIDE A DIFFERENT COLOR IF THEY ARE BOUNDARYS

for nel=1:nelem

for i=1:N

X(i) = XY(nodes(nel,order(i)),1);

Y(i) = XY(nodes(nel,order(i)),2);

end

% PLOT ONLY THE FIRST 9 POINTS OF X,Y (NODE 1 IS INCLUDED TWICE TO "CLOSE

% THE LOOP"

plot(X,Y)

% THE ELEMENT NUMBER IS SHOWN AT THE LOCATION OF NODE 9 (ESSENTIALLY,

% THE CENTER OF THE ELEMENT

text(X(N),Y(N),num2str(nel));

end

hold off

end

## Element:

function [ek,ef,node] = elem2D(XY,nel,k,dx,nodes,Case)

%Element stiffness and load matrices

%[ek][U1 U2 U3 U4]' = [f1 f2 f3 f4]'

%Case 1 -2nd Order Negative Case

%Case 2 -2nd Order Positive Case

%Case 3 -4th Order Negative Case

%Case 4 -4th Order Positive Case

%horiz\_vert-denotes values that correspond to nodes above/below or

% right/lift of node i,j in the stiffness matrix

%diagonal -refers to values corresponding to nodes diagonal of node i,j in

% the stiffness matrix

%center -denotes the value corresponding to node i,j in stiffness matrix

%RHS -denotes right hand side of fi

%Equations for each case are subject to depending on the differential

%equation. This code reflects +-(Uxx+Uyy)+k^2\*U=k^2\*x

switch Case

case 1 %2nd Order Negative Case

%define molecule values

horiz\_vert = -1;

center = (k^2\*dx^2) + 4;

RHS = k^2\*dx^2;

%elemental stiffness matrix

ek = [center, 2\*horiz\_vert,0,2\*horiz\_vert;...

2\*horiz\_vert,center,2\*horiz\_vert,0;...

0,2\*horiz\_vert,center,2\*horiz\_vert;...

2\*horiz\_vert,0,2\*horiz\_vert,center];

%elemental load

ef = [RHS\*XY(nodes(nel,1),1);...

RHS\*XY(nodes(nel,2),1);...

RHS\*XY(nodes(nel,3),1);...

RHS\*XY(nodes(nel,4),1)];

%define node numbers to be called later in assembly

node = [nodes(nel,1);nodes(nel,2);nodes(nel,3);nodes(nel,4)];

case 2 %2nd Order Positive Case

%define molecule values

horiz\_vert = 1;

center = (k^2\*dx^2) - 4;

RHS = k^2\*dx^2;

%elemental stiffness matrix

ek = [center, 2\*horiz\_vert,0,2\*horiz\_vert;...

2\*horiz\_vert,center,2\*horiz\_vert,0;...

0,2\*horiz\_vert,center,2\*horiz\_vert;...

2\*horiz\_vert,0,2\*horiz\_vert,center];

%elemental load

ef = [RHS\*XY(nodes(nel,1),1);...

RHS\*XY(nodes(nel,2),1);...

RHS\*XY(nodes(nel,3),1);...

RHS\*XY(nodes(nel,4),1)];

%define node numbers to be called later in assembly

node = [nodes(nel,1);nodes(nel,2);nodes(nel,3);nodes(nel,4)];

case 3 %4th Order Negative Case

%define molecule values

center = 40 + 8\*k^2\*dx^2;

diagonal = -2;

horiz\_vert = k^2\*dx^2 - 8;

RHS = 12\*k^2\*dx^2;

%elemental stiffness matrix

ek = [center, 2\*horiz\_vert,4\*diagonal,2\*horiz\_vert;...

2\*horiz\_vert,center,2\*horiz\_vert,4\*diagonal;...

4\*diagonal,2\*horiz\_vert,center,2\*horiz\_vert;...

2\*horiz\_vert,4\*diagonal,2\*horiz\_vert,center];

%elemental load

ef = [RHS\*XY(nodes(nel,1),1);...

RHS\*XY(nodes(nel,2),1);...

RHS\*XY(nodes(nel,3),1);...

RHS\*XY(nodes(nel,4),1)];

%define node numbers to be called later in assembly

node = [nodes(nel,1);nodes(nel,2);nodes(nel,3);nodes(nel,4)];

case 4 %4th Order Positive Case

%define molecule values

center = 8\*k^2\*dx^2 -40;

diagonal = 2;

horiz\_vert =8 + k^2\*dx^2;

RHS = 12\*k^2\*dx^2;

%elemental stiffness matrix

ek = [center, 2\*horiz\_vert,4\*diagonal,2\*horiz\_vert;...

2\*horiz\_vert,center,2\*horiz\_vert,4\*diagonal;...

4\*diagonal,2\*horiz\_vert,center,2\*horiz\_vert;...

2\*horiz\_vert,4\*diagonal,2\*horiz\_vert,center];

%elemental load

ef = [RHS\*XY(nodes(nel,1),1);...

RHS\*XY(nodes(nel,2),1);...

RHS\*XY(nodes(nel,3),1);...

RHS\*XY(nodes(nel,4),1)];

%define node numbers to be called later in assembly

node = [nodes(nel,1);nodes(nel,2);nodes(nel,3);nodes(nel,4)];

end